Software Technology for Adaptable, Reliable Systems (STARS)

Submitted to:
Electronic Systems Division:
Air Force Systems Command, USAF:
Hanscom AFB, MA 01731-5000:

Contract No:
F-19628-88-D-0028

CDRL 00850
Peer Review Capability Description

February 2, 1990

The Boeing Company
Space and Defense Group
Boeing Aerospace and Electronics
P.O. Box 3999
Seattle, Washington 98124

Approved for public release - distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	02-FEB-90	

4. TITLE AND SUBTITLE
Peer Review Capability Description

5. FUNDING NUMBERS
C: F19628-88-D-0028

6. AUTHOR(S)
Margaret J. Davis

TA: BR-40

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Boeing Company
Boeing Aerospace and Electronics Division
Systems and Software Engineering
P.O. Box 3999
Seattle, Washington 98124

8. PERFORMING ORGANIZATION REPORT NUMBER
D-613- 20850

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ESD/AVS
Bldg. 17-04
Room 113
Hanscom Air Force Base, 01731-5000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release - distribution unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(Maximum 200 words)*

This document describes the prototype peer revidw capability that was developed as part of the BR40 Repository Integration task. The peer review capability uses a public domain structure editor to support the peer review of Ada source code. The capability includes an Ada program to translate Ada source in ASCII text to the intermediate form used by the editor. The editor supports embedding peer comments in the Ada source in a form that is acceptable to an Ada compiler and processable by a SGML parser.

Besides describing the source peer review capability, this document contains recommendations for use on a repository and for productization.

14. SUBJECT TERMS
Keywords: STARS
peer review
structure editor

15. NUMBER OF PAGES
23

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	None

Peer Review Capability Description

Prepared by

Margaret J. Davis
Chief Programmer (R40)

Reviewed by

James E. King
System Architect

Reviewed by

John M. Neorr
Development Manager

Approved by

William M. Hodges
STARS Program Manager

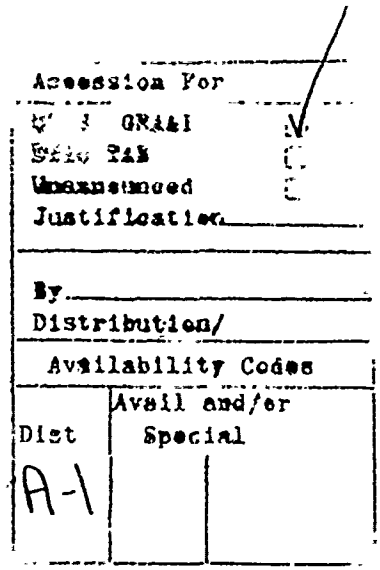
ABSTRACT

This document describes the prototype peer review capability that was developed as part of the BR40 Repository Integration task. The peer review capability uses a public domain structure editor to support the peer review of Ada source code. The capability includes an Ada program to translate Ada source in ASCII text to the intermediate form used by the editor. The editor supports embedding peer comments in the Ada source in a form that is acceptable to an Ada compiler and processable by a SGML parser.

Besides describing the source peer review capability, this documents contains recommendations for use on a repository and for productization.

KEYWORD STRINGS

peer review
structure editor



PREFACE

This document describes the fourth and final version in the series of prototypes of a peer review editor for Ada source code.

Note: The contents of this document are available in electronic form from a STARS repository.

TABLE OF CONTENTS

1. SCOPE

The purpose of this document is to describe the prototype peer review capability for Ada source code that was developed as part of BR40 Repository Integration. Besides describing the operation of the prototype, this document contains information related to installation on a repository and recommendations for further work.

One objective of the STARS Program is to institutionalize peer review of source code and technical documentation as part of the system development process. This prototype tool tests the concept of a folded tree editor to support on-line peer review of Ada source code.

The capability, named AdaPEERmacs was developed from a public domain editor known as GNU Emacs [GNU]. The tree editing functions were obtained from the University of Illinois. Basic descriptions and motivational material for tree rather than language-sensitive or syntax-directed editing are covered by [TREE].

Section 2 lists referenced documents; Section 3 describes the prototype operation in the form of a user guide; and, Section 4 contains recommendations.

2. REFERENCED DOCUMENTS

This section lists all documents referenced herein.

[GNU] "GNU Emacs Manual", Fifth Edition, Emacs Version 18, October 1986, Richard Stallman.

[TREE] "The TREEMACS tree editor and its applications", Computer Science Department, University of Illinois at Urbana-Champaign, Samuel Kamin and David Hammerslag, Draft January 5, 1989.

[2nd] "Ada as a second language", Norman H. Cohen, McGraw-Hill Book Company, 1986.

3. USER GUIDE TO ADAPEERMACS OPERATION

3.1 Operational Concept

3.1.1 Basic Notions

The basic concept underlying AdaPEERmacs is the notion of maintaining the Ada source code and peer commentary in a tree-based intermediate representation. The intermediate representation stores nodes as a single-rooted pre-order tree. Each node may have any number of child subtrees and represents either an Ada language construct or a peer commentary construct.

The display that the user sees uses indentation to indicate subtrees. Constructs with the same amount of indentation are all siblings or cousins of siblings. For instance, in the code fragment show below, the "with" and "procedure" constructs are siblings sharing a "comment" construct as their parent. The "while" construct is a cousin to "begin -- Line_Sum" since their parents ("begin -- Sum_ASCII_Codes" and "function" construct) are siblings.

```
-- This is page 458 from Ada as a [2nd] Language
  with Basic_IO;  use Basic_IO;
  procedure Sum_ASCII_Codes is
    Sum : Natural := 0;
    Length : Natural;
    Line : String(1..132);
    function Line_Sum (Line : IN String) return Natural is
      Result: Natural := 0;
      begin  -- Line_Sum
        for i in Line'Range loop
          Result := Result + Character'Pos(Line(i));
        end loop; -- for loop
        null;
      --:-)
      return Result;
    end Line_Sum;
    begin  -- Sum_ASCII_Codes
      while NOT End_Of_File loop
        Get_Line(Line, Length);
        Sum := Sum + Line_Sum(Line=>Line(1..Length));
      end loop; -- while loop
      Put(Sum);
      New_Line;
    --:-)
  end Sum_ASCII_Codes; -- procedure body
```

The major difference between a syntax-directed editor and AdaPEERmacs is that AdaPEERmacs does not require that the tree be syntactically and semantically correct Ada. The construct insertion functions support entering the correct syntax because they use a strategy of user prompting and template insertion, but they do not enforce Ada syntax. Syntactic and semantic checks are user initiated by invoking a command that sends an ASCII representation of the tree to an Ada compiler. This is a compromise between the minimal support that an experienced Ada programmer would desire in an editor and the maximal support that a novice Ada programmer would receive from a syntax-directed editor.

The notion of a folded display is also critical to comprehending what AdaPEERmacs puts on the screen. The AdaPEERmacs display may hide subtrees just as if someone folded a piece of a paper to cover up portions of printed material. The text below is a folded version of the code fragment previously displayed. The text for "function Line_Sum" and the statements following "begin -- Sum_ASCII_Codes" have been folded out of view.

```
-- This is page 458 from Ada as a [2nd] Language
 with Basic_IO;  use Basic_IO;
 procedure Sum_ASCII_Codes is
   Sum : Natural := 0;
   Length : Natural;
   Line : String(1..132);
   function Line_Sum (Line : IN String) return Natural is
   end Line_Sum;
   begin  -- Sum_ASCII_Codes
   --:-)
 end Sum_ASCII_Codes; -- procedure body
```

Storing the Ada program as nodes also permits AdaPEERmacs to maintain other information about the source code besides the type of construct each node represents. Since one objective for this prototype is support of peer review, AdaPEERMACS also stores the login name of the person who created the node, the login name of the person who last modified the node, the date that the node was created, and the date that the node was last modified. This administrative information is automatically displayed for peer comments but not for Ada constructs. However, the date information can be used to selectively display either Ada or peer commentary nodes that have been entered or modified after a specified date. Perhaps this administrative information will be used in a future configuration control or repository asset management application.

This extra information AdaPEERmacs is maintaining can be used to select to what is displayed on the screen. In the current version, there are functions that permit the user to selectively display (1) all nodes, which is the default; (2) only Ada nodes; (3) only Ada PDL nodes; (4) only peer commentary; (5) only nodes that have modified dates different than their creation dates; or (6) only nodes whose creation or modification

date is later than a user-specified date. Nodes not selected for display appear as blank lines. This means that the tree-point cursor (and text cursor) will seem to stop at random spots when cursor motions are requested. For this reason, the bottom line of the AdaPEERmacs display gives information about the node at the current position.

Furthermore, since the GNU Emacs editor upon which AdaPEERmacs is based can display separate windows, AdaPEERmacs also provides functions to automatically split the current window into two facing windows with different selective displays. (Facing windows means that the two displays appear as left-right facing pages in a book.) Commands that invoke tree cursor motion will automatically invoke cursor motion in all facing windows so that the pages will stay line-aligned. However, this line alignment will be lost if tree-editing commands are invoked in any facing window. Because of the control that the Emacs editor exerts over display of updated buffers, the tree cursor position will always remain correct although the facing display may not be correct. The display can be forced to stay synchronized with the tree cursor by requiring keyboard input from the user as each facing display is updated. The current prototype does not force synchronization.

The text below is an example of two facing windows where the left side displays only Ada constructs and the right side displays only peer commentary. (Note example contains "..." where elision has been used in order to fit the display on the page.)

```
procedure Sum_ASC...          |
Sum : Natural := 0;           |
                              |--|Peer_comment>
                              |-- This is a sample comment
                              |-- for demonstratio. purposes
                              |--|Created_By> mjdavis
                              |--|Created_On> Mon Dec 18...
                              |--|Last_Modified_By>mjdavis
                              |--|Last_Modified_On> Mon Dec 18...
                              |--||Peer_comment>
Length : Natural;             |
Line: String(1..132);         |
function Line_Sum...          |
end Line_Sum;                 |
```

3.1.2 Typical Edit Cycle

The next two sections describe editing starting with a blank display and starting with a non-blank display. The symbols "C-" before a character means the user holds down the control key while entering the next character. The symbols "M-" means the user

enters the escape key.

Just as normal editors maintain a current cursor position for the text being entered, AdaPEERmacs also maintains a position for the current tree node. The current text cursor position is known as point and the current tree position is known as tree-point.

Each node of the tree may be thought of as a separate piece of text with its own point. GNU Emacs refers to these pieces of text as buffers. GNU Emacs commands that position the cursor and enter text within a buffer are scoped within the text of the node. For example, the command that positions the text cursor at the beginning of a buffer will position the text cursor at the beginning of the node's text. Two portions of a node's display (its label and end label) are not within the scope of the node's text. These are synthesized by AdaPEERmacs from the node's level within the tree (this allows correct indentation), its construct name, and possibly administrative information or information derived from its parent node. The most basic construct is ada-statement, which has a label consisting only of indentation spacing and a close label of ";". Other constructs have more elaborate labels or close labels. For instance, the label for peer comments is "--|Peer_comment>" and its close label contains all the administrative information.

3.1.2.1 Blank Display

Starting with a blank display, the goal is to insert Ada constructs to create an Ada program.

1. Since the tree representation starts with a single root, it is convenient to begin by inserting an Ada comment. This is accomplished by entering the key sequence "C-cC-t-" or "M-x ada-comment". AdaPEERmacs will create a node labeled "ada-comment", will update the display with 3 lines starting with "--" and then position the text cursor at the middle line after the "--". The user may then enter text for the comment.

2. The next step is to enter an Ada construct. The key sequence specifying a template ("C-cC-t"). may be entered followed by the key indicating which template is desired or the key sequence "M-x" followed by the desired construct name may be entered. AdaPEERmacs will prompt for the information needed to fill in the template and will ask if the template is to be inserted as nested or not. Responding "y" will create a child node and responding "n" will create a sibling node. Since the tree only contains a root node, respond "y" to create a child node. Note that one template may generate multiple tree nodes. For instance, the ada construct ada-function-body will insert a node for itself, an ada-declare node for its return value declaration, an ada-begin-noend node, and ada-statement node whose text is "return RESULT_NAME" and an ada-statement node whose text is "null". The template inserts these as children or sibling nodes as is appropriate to represent the control or declaration flow

each construct represents.

3. All the remaining steps repeat the template invocation (step 2). There are key sequences for positioning the current tree-point where the user desires to insert the template. "C-cC-yu" moves up the tree via children and siblings; "C-cC-yd" moves down the tree via children and sibling (depth-first motion). Both of these sequences may fold/unfold the display. The sequences "C-cC-yC-u" and "C-cC-yC-d" move more quickly by only moving sibling to sibling (breadth-first motion).

4. The editor may be exited and the file saved by entering C-xC-c.

3.1.2.2 Non-blank Display

Starting with a non-blank display of Ada source, the goal is to enter peer review comments.

1. The first step is to position the tree-point at the construct to which the comment will apply. "C-cC-yu" moves up the tree via children and siblings; "C-cC-yd" moves down the tree via children and sibling (depth-first motion). Both of these sequences may fold/unfold the display. The sequences "C-cC-yC-u" and "C-cC-yC-d" move more quickly by only moving sibling to sibling (breadth-first motion). At the beginning of an editing session the tree-point is positioned at the root. The last line of the display lists the construct of the tree-point and its creation date.

2. The next step is to enter an ada-peer-comment construct. The key sequence specifying a template ("C-cC-t?") may be entered or the key sequence "M-x ada-peer-comment" may be entered. AdaPEERmacs will put the label and close label for an ada-peer-comment on the display and position the text cursor after a middle line contain "--". The text for the comment can now be entered. Every time a "Return" key is entered, the text will be updated with a newline containing the necessary amount of indentation space and a "--". If the tree-point where the comment is to be inserted is an ada-peer-comment or already has a child that is an ada-peer-comment, the template will insert a construct known as ada-peer-followon. This allows a history of comments to be collected.

3. All the remaining steps repeat the steps 1 and 2.

4. The editor may be exited and the file saved by entering C-xC-c.

3.2 Functionality

3.2.1 Tree editing and Motion Functions

The tree editing commands and motion functions that can be invoked by entering CTL-C followed by CTL-Y followed by the key identifying the function are (note that they can also be invoked by M-x followed by function name):

```
"p"    !parent -- Move tree cursor (tree-point) to parent
node

"u"    !up -- Move tree cursor (tree-point) to next node up
in tree (left sibling or parent node)

"C-u" !up-condensed -- Move tree cursor (tree-point) to
parent or left sibling node

"C-d" !down-condensed -- Move tree cursor (tree-point) to
right sibling node

"d"    !down -- Move tree cursor (tree-point) to next node
down in tree (child or right sibling)

"c"    !child -- Move tree cursor (tree-point) to child node

"l"    !left -- Move tree cursor (tree-point) to left
sibling node

"r"    !right -- Move tree cursor (tree-point) to right
sibling node

"C-f" !char-right -- move text cursor 1 character to right
in current node

"C-b" !char-left -- move text cursor 1 character to left in
current node

"c-p" !previous-line -- move text cursor to preceding line
in current node

"c-n" !next-line -- move text cursor to following line in
current node

"C-l" !redraw-display -- redraw the display
```

"T" !goto-root -- move tree cursor (tree-point) to tree root node

"e" !tree-edit-current-node -- edit the subtree rooted at the current node in a separate window (exit by entering M-C-c)

"g" !goto-node -- go to a specific node

"#" !set-tree-mark -- set the value of mark

"@" !mark-current-node -- remember this node as mark

"M-u" !update -- update the tree with all changes

"E" !show-all -- Make the display show all the nodes

"A" !show-ada -- Make the display show only Ada nodes

"P" !show-adl -- Make the display show only Ada pdl nodes

"C" !show-comments -- Make the display show only peer comments

"M" !show-modified -- Make the display show only nodes whose modified date is not the same as created date

"D" !show-after -- Make the display show only nodes whose created or modified date is after a specified date

"FE" !face-all -- Split the current window into two facing separate windows and display all the nodes in the new window (exit by entering M-C-c)

"FA" !face-ada -- Split the current window into two facing separate windows and display only ada nodes in the new window (exit by entering M-C-c)

"FP" !face-adl -- Split the current window into two facing separate windows and display only ada PDL nodes in the new window (exit by entering M-C-c)

"FC" !face-comments -- Split the current window into two facing separate windows and display only peer commentary nodes in the new window (exit by entering M-C-c)

"FM" !face-modified -- Split the current window into two facing separate windows and display only modified nodes in the new window (exit by entering M-C-c)

"FD" !face-after -- Split the current window into two facing separate windows and display only nodes created or modified after a specified date in the new window (exit by entering M-C-c)

Other motion and tree editing commands that are only invoked by M-x followed by function name are:

!delete-tree -- delete the current node and its subtree

!read-over-root -- replace the root node with the specified tree from a file

!write-tree -- write the entire tree to disk in both an ascii Ada format and its intermediate AdaPEERmacs format

!read-child -- read a tree from a file and add it as a child subtree of the current node

!write-subtree write the tree rooted at the current node to disk in both an ascii Ada format and its intermediate AdaPEERmacs format

!tree-kill -- delete the subtree rooted at the current node, saving it for later yanking.

!tree-yank -- add a copy of the most recently killed tree as a subtree to the current node

Node text motion commands that may be invoked by the following identifying key sequence (or by entering M-x followed by function-name) are:

"M-<" !beginning-of-node -- move text cursor to beginning of node's text

"M->" !end-of-node -- move text cursor to end of node's text

"C-b" !backward-char -- move text cursor 1 character to left

"C-f" !forward-char -- move text cursor 1 character to right

"C-p" !previous-line -- move text cursor to preceding line

"C-n" !next-line -- move text cursor to following line

"C-a" display-pretty-printed -- show the entire tree as an ascii file in a separate window

3.2.2 Other Useful Functions

Other useful functions that are invoked by entering a specific key sequence or by entering M-x followed by function-name are:

"C-a" display-pretty-printed -- show the entire tree as an ascii file in a separate window

"C-g" quit -- discard/abort current command

"C-hC-h" help-for-help -- display an explanation of help functions available

"C-xo" other window -- move to the next window (that's a letter o not a number 0)

"C-x1" delete-other-windows -- close all windows but the window where the cursor is located (that's a number 1 not a letter l)

"C-xC-f" find-file -- find a file (and open it for display)

"C-xC-c" save-buffers-kill-emacs -- offer to save each modified file and then exit.

3.2.3 Templates

All templates are invoked by entering CTRL-c followed by CTRL-t followed by template identifier key.

"h" ada-header

"-" ada-comment

"C-a" ada-array

"A" ada-accept

```
"a"   ada-stmt
"b"   ada-exception-block
"D"   ada-declare-block
"d"   ada-declare-stmt
"c"   ada-case
"C-e" ada-elsif
"e"   ada-else
"C-k" ada-package-spec
"k"   ada-package-body
"C-n" ada-entry
"C-p" ada-procedure-spec
"p"   ada-subprogram-body
"C-f" ada-function-spec
"f"   ada-for-loop
"i"   ada-if
"l"   ada-loop
"o"   ada-or
"C-r" ada-record
"C-s" ada-subtype
"s"   ada-select
"S"   ada-separate
"C-t" ada-task-spec
"t"   ada-task-body
```

```
"C-y" ada-type

"C-u" ada-with-use

"C-v" ada-private

"C-w" ada-when

"w"   ada-while-loop

"C-x" ada-exception

"x"   ada-exit

"|"   ada-pdl

"?"   ada-peer-comment

"!"   ada-peer-response
```

4. RECOMMENDATIONS

4.1 Lessons Learned

4.1.1 Regarding GNU Emacs

Our expectations regarding the extensibility and power of GNU Emacs were fulfilled. Adding new commands, binding commands to keystroke combinations, replacing commands, and adding help for new commands are all relatively simple modifications. However, the buffer/window metaphor that GNU Emacs uses is not as conducive to facing page editing as we hoped, but we used Emacs' extensibility to overcome most of the problems.

The remaining problem is that the amount of control GNU Emacs exerts over display updating interferes with keeping the facing displays synchronized as a user scrolls. The display may not be updated to reflect the actual cursor position. Requiring keyboard input from the user in the form of question-response can force the update, but other informational messages may mask the questions. The user can also force an update by switching to the facing display and requesting a display update. Perhaps it would be better to abandon the concept of automatically keeping the facing displays synchronized and leave display scrolling entirely to the user. This solution is the one used by high performance workstations that provide multiple, simultaneous editing sessions on the same file by the same user.

4.1.2 Regarding Structure Editing

The Emacs extensions we used from software furnished by the University of Illinois [TREE] provided the basic capability for editing the Ada source code as a tree of Ada language constructs. Since checking conformance of the Ada source code to correct syntax and semantics is not part of structure editing, selecting the optimum level of abstraction for the Ada language constructs was the most difficult task. We chose the level of abstraction that is used by prettyprinters (flow of control and declarations). Whether this is optimum or not will have to be tested by deployment.

Structure editing has some similarity with context sensitive editing (such as DIANA supports) in that the editor is maintaining information about the structure. The software provided by the University of Illinois enabled us to also associate administrative information with each construct represented in the tree. Thus, we were able to customize the editor to selectively display only certain constructs (Ada, Ada comments, peer comments, etc) and to selectively display by creation and modification dates and user identification. This information would be relevant to tools that were responsible for access control and configuration or asset management. If the STARS infrastructure were to be standardized on DIANA or IRIS as an intermediate representation, it would be useful if the editor supporting the intermediate representation also automatically collected the administrative information.

4.1.3 Regarding Editor Presentation

AdaPEERmacs implements several techniques in its display:

explicit structure presentation,
selective presentation,
folded page presentation, and
facing page presentation.

We mimicked the indentation style of prettyprinters to reveal the structure of the Ada source code. It seemed most comfortable for an experienced programmer. This choice also let us partially support some of the Ada syntactic rules as an aid to novice programmers while not constraining experts. A person entering code should be able to concentrate on the "what" of the program being built rather than on the "what" of the Ada syntax. Most of the syntactic support is being provided by the start and end labels provided for each construct. Beginning keywords, such as "loop", are usually assigned as the start label. The Ada statement terminating ";" along with ending keywords are usually assigned as the end label. The only subtlety in creating this type of indented structure presentation is the correct display of nested control such as loops within loops.

Since structure editing requires an intermediate form wherein information about the structure is stored, selective presentation is easily implemented. The major decision relative to selective presentation is the treatment of material not to be displayed. We chose to use blank lines and provide feedback to the user as to cursor position. This is particularly useful when the cursor is stopped over blanked out areas. This choice was made in order to facilitate facing page presentation.

The folded page presentation is also a direct consequence of tree editing. It can be disconcerting at first when a command to scroll radically changes the display. However, this confusion is easily removed when the folded concept is explained. We increased the utility of the folded page presentation by providing two sets of scroll commands. One sets scrolls the tree in a depth-first fashion (unfolds) and the other set scrolls the tree in a breadth-first fashion (does not unfold). This gives the user a choice in the level of detail displayed. Folded page presentation should be even more useful in document preparation and review because it effectively lets the user scroll an outline of the document.

We used the capabilities of GNU Emacs for splitting a terminal display into several screens to implement the facing page presentation. Facing page presentation is basically a technique for visually synchronizing selective displays. It is most effective when one page displays all material and the other displays selected material. We succeeded in automatically keeping the displays synchronized most of the time.. More in-depth knowledge of GNU Emacs I/O handling would be required to achieve synchronization all the time. The blank line style we chose for selective display works

well for facing page presentation of Ada source on one side and peer comments on the other because the comments are only associated with a few lines of text. (Refer to page 10 for an example of this style.) This may not be satisfactory for peer comments in documents. A more elaborate scheme that put the peer comment on the facing page line that is parallel to the top of the corresponding document element (section, chapter, paragraph, etc.) would be better. It would also insert blank lines after the document element if the peer comment required more lines than the document element used. This style is illustrated below:

```
Section x.y Recommendations     |
                                |
In light of the new...          |These are not new features.
we recommend purchase.          |They are available from Ajax
                                |Software at a cheaper price.
                                |
We also recommend purchase of   |In-house maintenance would be
a maintenance contract from     |cheaper and as good.
the same company.               |
```

4.2 Suggestions for Repository Installation

The prototype currently is based on a BSD 4.2 version of GNU Emacs. Installation and trial use at the STARS repository would require porting the prototype to an existing VMS GNU Emacs implementation. It would also be useful to "bytecompile" as much of the modifications as possible to improve its performance. (Most of the modifications are currently being "interpreted" by the GNU editor.)

It would also be desirable to put a user-friendly "shell" around the editor invocation that would make the translation to intermediate form and back again invisible to a user. Some of this could be provided by GNU Emacs modifications and some by an operating system command script.

4.3 Suggestions for Further Work

More work is needed in replacing the standard editor commands to correctly function within a tree node's text. More work is also needed in checking that the deletion and copying of subtrees works correctly. Presumably these functions were provided by the code received from the University of Illinois, but we have not tested them.

A version that provided peer review entry for SGML-tagged documents would also be useful. Such an editor would support creation of SGML-tagged documents while hiding the cumbersome tags from the user's view. The start and end labels could be used for a limited type of WYSIWYG ("What You See Is What You Get") presentation. Also, it would be beneficial to implement the more elaborate form of facing page presentation discussed in section 4.1.3 above.

The *Boeing* Company

ACTIVE SHEET RECORD

SHEET NO.	REVLTR	ADDED SHEETS				SHEET NO.	REVLTR	ADDED SHEETS			
		SHEET NO.	REVLTR	SHEET NO.	REVLTR			SHEET NO.	REVLTR	SHEET NO.	REVLTR
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
14											
16											
17											
18											
19											
20											
21											
22											
23											

THE *BOEING* COMPANY

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVAL